Mechatronics Software Lead Report

EE30220

12371

Word Count: 1156

Integration process

At the start of the project I was assigned as the software lead. Having not programmed for a while I begun my approach by drawing a flow diagram and outlining key objectives that I wanted to align with throughout the design. Using the brief I picked out key events that happen in the whole process and drew them as seen in figure 1.

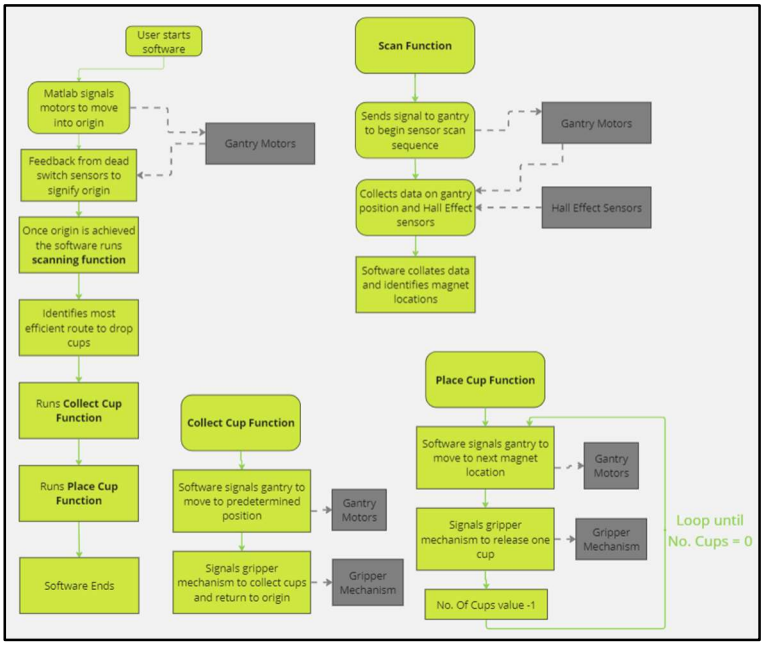

Figure 1 – Flow diagram created for the software.

Creating the flow diagram allowed me to understand which parts of the code I could work on at different stages of design process. In addition to the flow diagram, I creaded a specifications table, Table 1, which I refered back to through out the design process. I began by writing psudo-code for each part of the process and then adapt it when the electrical and mechanical parts of the project were ready.

No.	Specification	Wish/Demand	Testing	Expected Outcome
1.1	Accurately Collect and store data	D	Compare data from runs and check after runs	Data regarding gantry position will be accurate and stored on the computer
1.2	Identify magnet locations	D	Comparing softwares idintified magnet locations to real life	Software will identify locations with high degree of accuracy
1.3	Tidy and Efficient code	W	Visual Inspection	Use of Functions, Arrays and loops to shorten code
1.4	Fail safe incase of error	W	Software doesn't get stuck in loops	Software has unable to be stuck in loops
1.5	Efficient code	W	Check if code can be shortened in any way	Code will be efficient and have no unnecessary sections

Table 1 – Specification table for software.

I started by learning to control the Gantry as this did not require either the mechanical or electrical parts of the projects. I begun by researching the different options on how to program the stepper motors to move. PWM and Tone were the two options that I found, each with their advantages and disadvantages. PWM allowing both motors to move simultaneously yet being slow compared to Tone which had a significantly higher speed but could only move one motor at a time. As I was unsure on what the other member of the group prefered, I created two functions which assigned the stepper motors to either Tone or PWM.

```
function ToneMode(ard)
Pulsey = 'D4';
Pulsex = 'D8';
configurePin(ard,Pulsex,'Tone');
configurePin(ard,Pulsey,'Tone');
disp("Tone Mode");
end
```

Figure 2 – Tone assigning function.

```
function PWMMode(ard)
Pulsey = 'D4';
Pulsex = 'D8';
configurePin(ard,Pulsex,'PWM');
configurePin(ard,Pulsey,'PWM');
disp("PWM Mode");
end
```

Figure 3 – PWM assigning function.

Electical Integration Testing

To integrate the sotware with the electrical lead, the first step was to test their up op-amp circuit. To do this I researched how to read a voltage signal through an arduino and plot it on a graph. Initially beginning with Figure 5 which allowed us to see how close to the magnets our hall effects would have to be to pick up the location. Following on from this I also programmed a script that displayed the hall effect sensor strength as a radius of a circle (Figure 4). This was incase we decided to use a locating method which incorperated 3 circles in a triangle pattern. We ended up not using this method due to the complexity and risks of multiple recordings of the same magnet. If I was to progress the code further, I would revisit this idea and attempt to get it working.

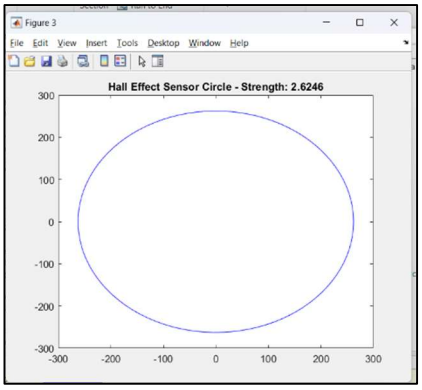

Figure 4 – Hall effect sensor strength displayed as circle.

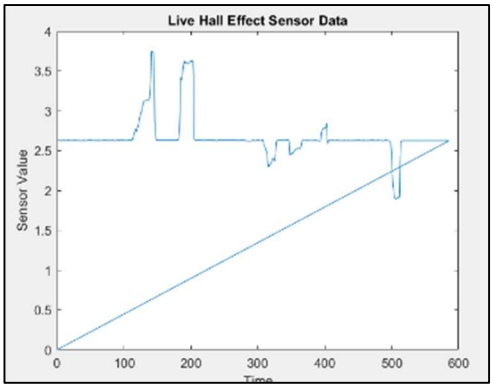

Figure 5 – Hall effect sensor strength displaying output voltage.

Once we were able to get consistent readings from the op-amp, we worked on discussing wether we would increase the amplitude on the software side or by increasing the output voltage on the electrical side. We decided to do this on the software side. By taking an initial background noice

reading for each of the sensors, this was then used to off set the voltage about 0. A variable multiplier was then added to help the magnets stand out when a change in voltage occurred.

The next step was to integrate this into the scanning script I wrote and to test it with 3 hall effect sensors. This was done through the use of recurring loops that took recordings at intervals as the gantry moves.

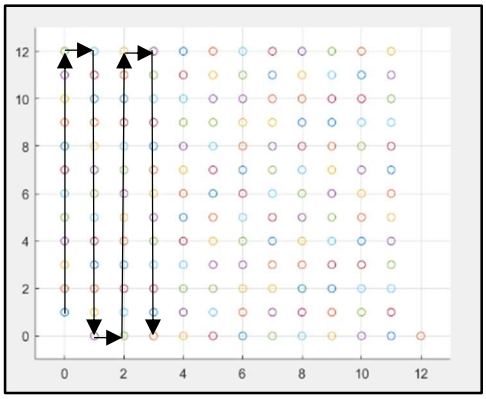

Figure 6 – Scanning pattern and intervals.

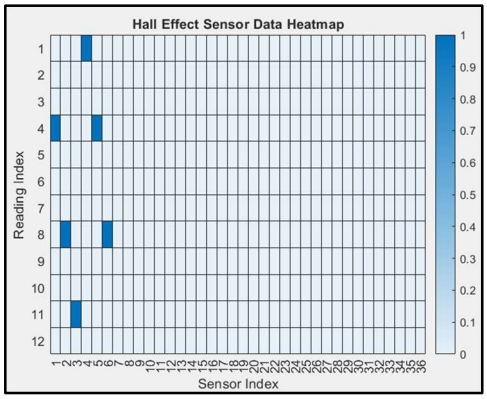

Figure 7 – Hall effect sensor matrix in a heatmap.

Figure 6 showing the path of travel and each data point being a recording which records 3 values and sets them in a matrix. The scanning script would create a matrix for the scanned area, adjust the signal for any noise and then filter any peaks assigning a 1 for any values over a certain amount and 0 for all else. Allowing a heatmap to be plotted showing the locations of the magnets. When testing the scanning script on the gantry I found it would overshoot the readins and record double sometimes. To resolve this I worked through my code by hand recording the data at each point until I found the problem and fixed it.

t	step	xcord	ycord	interval	numreadings	j	matrix x	matrix y
0	0	0	0	0	12			
				0				
	1		1			1	1	1
						2	1	2
						3	1	3
	2		2			1	2	1
						2	2	2
						3	2	3
	3		3			1	3	1
						2	3	2
						3	3	3
	12		12			1	12	1
						2	12	2
						3	12	3
	1	1						
				1				
	0							
	12							
	11		11			1	11	4
						2	11	5
						3	11	6
	10		10			1	10	4
						2	10	5
						3	10	6
	0		0					

Table 2 – Testing Scanning Script Data.

We encountered some problems whilst testing which led to more magnets being detected then were actually present. To solve this, I edited the script to also out put a surf of the matrix keeping the raw data. This allowed us to see if it was a software problem, electrical problem or an anomolous result. I also researched methods of merging any repeated detections within a certain distance of eachother in the matrix. A merging detections function was then added to the script and tested to try and avoid this problem.

Mechanical Integration Testing

To start integrating the mechanical and software aspects I created multiple scripts for testing the different types of motors; gantry, servo and DCDC (Figures 8 and 9). These scipts allowed us to test the different abilities of the different motors. Initially, We used a continuous servo motor when varying the z axis but found that a lot of overshoot occures when changing directions. As we needed to pick up and place cups at specific z heights we needed to change motors. This prompted the swap to a 180 degree Servo. With this we could more accurately control the hight of the mechanism and achieve repeated results.

```
clear ard;
clear servo;
ard = arduino();
pulsez = 'D22';
configurePin(ard,pulsez,'DigitalOutput');
servo = servo(ard, pulsez);

%up 0
writePosition(servo, 0);
pause(2)


%pause(1);
%writePosition(servo, 0.5);
clear servo;
%clear ard;
disp('done');
```

Figure 8 – Servo motor testing script.

```
clear;
% Create an Arduino object
ard = arduino(); % Replace 'COM3' with your Arduino port

% Define motor control pins
enablePin = 'D11'; % PWM pin for motor speed control
in1Pin = 'D13'; % Motor driver input pin 1
in2Pin = 'D12'; % Motor driver input pin 2

% Set up motor control pins
configurePin(ard, enablePin, 'PWM');
configurePin(ard, in1Pin, 'DigitalOutput');
configurePin(ard, in2Pin, 'DigitalOutput');

writePWMVoltage(ard, enablePin, 3); % Adjust voltage to control speed
writeDigitalPin(ard, in1Pin, 0); % Set direction (adjust for your setup)
writeDigitalPin(ard, in2Pin, 1); % Set direction (adjust for your setup)

pause(0.35); % Run the motor for 0.35 seconds
writePWMVoltage(ard, enablePin, 0); % Stop the motor
pause(2)

% Clear Arduino connection
clear ard;
```

Figure 9 – DCDC motor testing script.

Once mechanism was built we were able to use the testing scripts to get the exacts values needed and implement them into the scanning script. Using the DCDC motor testing script we also were able to find the timings needed to close and open the iris mechanism.

Conclusion and Reflection

Through out this project I wanted to reintroduce myself to software programming and Matlab. My goal was to produce an effective code which was also efficient. It was improtant to me to keep the code efficient and tidy, as this was a group project I wanted my code to be easily understood by the other members. Using variables, functions and loops throughout the design/coding process would allow me to achieve this. Overall, I believe that I was able to achieve this goal as my group members were able to take my code and test the system without me there. I do believe that there is a lot of improvement that I could work on, for example using global variable or a more efficient way of assigning the arduino pins. I also started the project by using individual scripts as appose to functions which delayed some progress. Next time I would consider working on a user interface so the software is easier to set up/understand. In addition to this, I would look into alternative filtering methods to find the magnets such as findPeaks as it wouldn't require as much adjustment of the raw data.

Code Appendix

```
clear ard;

% Defining and configuring the main Arduino pins
ard = arduino();
enablex = 'D45';
direcx = 'D43';
Terminalx = 'D39';
enabley = 'D53';
direcy = 'D51';
Terminaly = 'D47';
Pulsex = 'D41';
Pulsey = 'D49';
configurePin(ard,enablex,'DigitalOutput');
configurePin(ard,direcx,'DigitalOutput');
configurePin(ard,Terminalx,'DigitalInput');
configurePin(ard,enabley,'DigitalOutput');
configurePin(ard,direcy,'DigitalOutput');
configurePin(ard,Terminaly,'DigitalInput');


% Moving the gantry to the Origin position
OriginPlacement(ard);

% Start scanning
coordMatrix = ScanningScript(ard);

% Use the matrix from scan to place cups
CoordTest(coordMatrix,ard);

% Return to origin to end task
OriginPlacement(ard);
```

Script 1 - Main Script (Ran to start the test)

```
function OriginPlacement(ard)

% Configuring the relevant pins
enablex = 'D45';
direcx = 'D43';
Terminalx = 'D39';
enabley = 'D53';
direcy = 'D51';
Terminaly = 'D47';
Pulsex = 'D41';
Pulsey = 'D49';
configurePin(ard,enablex,'DigitalOutput');
configurePin(ard,direcx,'DigitalOutput');
configurePin(ard,Terminalx,'DigitalInput');
configurePin(ard,enabley,'DigitalOutput');
configurePin(ard,direcy,'DigitalOutput');
configurePin(ard,Terminaly,'DigitalInput');
configurePin(ard,Pulsex,'Unset');

% Start loop that moves infinitely until the kill switch is hit
disp('Moving to Origin')
while true
    writeDigitalPin(ard,direcy,0);
    playTone(ard,Pulsey,1500,0.2);
    Terminatey = readDigitalPin(ard,Terminaly);
    if Terminatey == 0;
       break;
    end
end
pause(0.5)
while true
    writeDigitalPin(ard,direcy,1);
    playTone(ard,Pulsey,200,0.2);
    Terminatey = readDigitalPin(ard,Terminaly);
    if Terminatey == 1;
       break;
```

```
    end
end

% Unsets pin so gantry can move in opposite direction
configurePin(ard,Pulsey,'Unset');

% Start loop that moves infinitely in other direction until the kill switch is hit
while true
    writeDigitalPin(ard,direcx,0);
    playTone(ard,Pulsex,1500,0.2);
    Terminatex = readDigitalPin(ard,Terminalx);
    if Terminatex == 0;
       break;
    end
end
pause(0.5)
while true
    writeDigitalPin(ard,direcx,1);
    playTone(ard,Pulsex,200,0.2);
    Terminatex = readDigitalPin(ard,Terminalx);
    if Terminatex == 1;
       break;
    end
end
disp('At Origin');
%clear;
```

Script 2 – Script that returns the gantry to the origin 0,0 position.

```
function coordMatrix = ScanningScript(ard)

% Assigns and configures the relevant variables and Arduino pins
clear s;
pulsez = 'D22';
configurePin(ard,pulsez,'DigitalOutput');
s = servo(ard, pulsez);
figure()
Pulsey = 'D41';
Pulsex = 'D49';
direcy = 'D43';
direcx = 'D51';
xcord = 0;
ycord = 0;

% Defining plots dimensions
xlim([-1 6]);
ylim([-1 13]);
grid("on");
hold on;

% Defining hall effect sensor matrix and configuring pins
numReadings = 6;
writePosition(s, 1);
pause(2)
hallMatrix = zeros(numReadings, 3*6);
sensorPins = {'A1', 'A2', 'A3'};
for i = 1:numel(sensorPins)
    configurePin(ard, sensorPins{i}, 'AnalogInput');
end

% Taking background noise to adjust hall effect readings
Halleffect1 = readVoltage(ard, sensorPins{1});
Halleffect2 = readVoltage(ard, sensorPins{2});
Halleffect3 = readVoltage(ard, sensorPins{3});
disp('Starting Scan')

% Starts scanning with the interval being number of lengths.
for interval = 0:1:5
    if mod(interval, 2) == 0;
        step = 0;
        ycord = 0;
        % The gantry moves up the Y axis
        writeDigitalPin(ard,direcy,1);
        playTone(ard,Pulsey,600,7)
        while true
```

```
            % Whilst it moves a lap this loop records data every 0.58 seconds
            step=step+1;
            ycord = ycord + 1;
            scatter(xcord,ycord);
            hallMatrix(step,1+(interval*3)) = ((readVoltage(ard, sensorPins{1})-Halleffect1)*5);
            hallMatrix(step,2+(interval*3)) = ((readVoltage(ard, sensorPins{2})-Halleffect2)*5);
            hallMatrix(step,3+(interval*3)) = ((readVoltage(ard, sensorPins{3})-Halleffect3)*5);
            pause(0.58);
            if step == 10;
                 break;
            end
        end

    else
        % The gantry moves in the negative Y direction
        writeDigitalPin(ard,direcy,0);
        playTone(ard,Pulsey,600,7);
        step = 10;
        while true
            % Whilst it moves a lap this loop records data every 0.58 seconds
            step=step-1;
            ycord = ycord - 1;
            scatter(xcord,ycord);
            hallMatrix(step,1+(interval*3)) = ((readVoltage(ard, sensorPins{1})-Halleffect1)*5);
            hallMatrix(step,2+(interval*3)) = ((readVoltage(ard, sensorPins{2})-Halleffect2)*5);
            hallMatrix(step,3+(interval*3)) = ((readVoltage(ard, sensorPins{3})-Halleffect3)*5);
            pause(0.58);
            if step == 1;
                break;
            end;
        end;
    end
    % The gantry then moves once in the positive x direction
    if interval < 5
    configurePin(ard,Pulsey,'Unset');
    writeDigitalPin(ard,direcx,1);
    playTone(ard,Pulsex,600,1.8);
    pause(1.8)
    xcord = xcord +1;
    scatter(xcord,ycord)
    configurePin(ard,Pulsex,'Unset');
    end
end

% Makes all the voltage readings positive values while saving the raw values separately
hallMatrixRaw = hallMatrix;
hallMatrix = abs(hallMatrix);

% Output a surf to analyse raw data and apply a filter to the positive data to identify the magnet
locations
figure()
surf(hallMatrixRaw);
hallMatrix(hallMatrix < 2) = 0;
hallMatrix(hallMatrix >= 2) = 1;

% Heat map is created
figure()
heatmap(hallMatrix);
title('Hall Effect Sensor Data Heatmap');
xlabel('Sensor Index');
ylabel('Reading Index');
colorbar;

% mergeDetections function is used to join any repeat readings of the same magnets
mergedHeatmap = mergeDetections(hallMatrix);

% The coordinate locations of the magnets are output in a separate matrix
[row, col] = find(mergedHeatmap == 1);
disp([row, col]);
coordMatrix = [col,row];
disp('Scan Complete')

end
```

Script 3 – Scanning Script that scans the area and outputs a matrix for locating the treasure.

```
function mergedHeatmap = mergeDetections(hallMatrix)
    % Function to merge nearby detection points in the heatmap
    threshold = 1;  % This value dictated how close the repeats had to be (Sensitivity)

    % Varibles to allocate linked and individual detected magnets
    labeledHeatmap = bwlabel(hallMatrix);
    uniqueLabels = unique(labeledHeatmap(:));

    % Create a new heatmap for merged data
    mergedHeatmap = zeros(size(hallMatrix));

    for i = 2:numel(uniqueLabels)
        % Extract the coordinates of the current detection point, calculate
        % the center and check for other repeats in the threshold
        [row, col] = find(labeledHeatmap == uniqueLabels(i));
        centroid = [mean(col), mean(row)];
        distances = sqrt((centroid(1) - col).^2 + (centroid(2) - row).^2);
        nearbyDetections = distances <= threshold;

        % Merge the repeats to one point
        mergedHeatmap(round(centroid(2)), round(centroid(1))) = 1;

    end
end
```

Script 4 – Merging script to reduce magnets being double detected.

```
function CoordTest(coordMatrix,ard)

% Assigns and configures the relevant variables and Arduino pins
enablex = 'D45';
direcx = 'D43';
Terminalx = 'D39';
enabley = 'D53';
direcy = 'D51';
Terminaly = 'D47';
Pulsex = 'D41';
Pulsey = 'D49';
configurePin(ard,enablex,'DigitalOutput');
configurePin(ard,direcx,'DigitalOutput');
configurePin(ard,Terminalx,'DigitalInput');
configurePin(ard,enabley,'DigitalOutput');
configurePin(ard,direcy,'DigitalOutput');
configurePin(ard,Terminaly,'DigitalInput');

% Calculate the distances to each coordinate
coordinates = coordMatrix;
disp(coordinates)
distances = sqrt(sum(coordinates.^2, 1));

% Sort the coordinates based on distances
[sorted_distances, idx] = sort(distances, 'descend');
sorted_coordinates = coordinates(:, idx);
flippedMatrix = flipud(sorted_coordinates);

% Move gantry to sorted coordinates
for i = 1:size(flippedMatrix,1)
    x_coord = flippedMatrix(i, 1);
    disp(x_coord)
    y_coord = flippedMatrix(i, 2);
    disp(y_coord)
    % Starts picking up the ‘I’ number cup
    PickUpCup(i,ard);
    % Move motors to the coordinates
    motorpick = 1;
    coord = x_coord-1;
    move_stepper_motor(motorpick,coord,ard); % Move motor 1 to x-coordinate
    motorpick = 2;
    coord = y_coord+1.5;
    move_stepper_motor(motorpick,coord,ard); % Move motor 2 to y-coordinate

    %Move to cup offset
    writeDigitalPin(ard,direcx,1);
```

```
    playTone(ard,Pulsex,600,1.22);
    pause(1.22)

    %Open Lense and drop cup
    OpenLense(ard);
    disp(i)
    pause(1);
end

end
```

Script 5 – Cup placing Function.

```
function PickUpCup(i,ard)

% Configure the relevant pins and variables
enablex = 'D45';
direcx = 'D43';
Terminalx = 'D39';
enabley = 'D53';
direcy = 'D51';
Terminaly = 'D47';
Pulsex = 'D41';
Pulsey = 'D49';
configurePin(ard,enablex,'DigitalOutput');
configurePin(ard,direcx,'DigitalOutput');
configurePin(ard,Terminalx,'DigitalInput');
configurePin(ard,enabley,'DigitalOutput');
configurePin(ard,direcy,'DigitalOutput');
configurePin(ard,Terminaly,'DigitalInput');


enablePin = 'D11'; % PWM pin for motor speed control
in1Pin = 'D13'; % Motor driver input pin 1
in2Pin = 'D12'; % Motor driver input pin 2
configurePin(ard, enablePin, 'PWM');
configurePin(ard, in1Pin, 'DigitalOutput');
configurePin(ard, in2Pin, 'DigitalOutput');

% Changes Z height and moves to the origin
pos = 0;
ServoMotor(ard,pos);
pause(1);
OriginPlacement(ard);
pause(0.5);

% Uses the i number to determine where the next cup is to pick up
if i <= 6
    configurePin(ard,Pulsey,'Unset');
    writeDigitalPin(ard,direcx,1);
    playTone(ard,Pulsex,1500,4.44);
    pause(4.44);
    configurePin(ard,Pulsex,'Unset');
    if i > 1
    writeDigitalPin(ard,direcy,1);
    playTone(ard,Pulsey,1000,(1.27*(i-1)));
    pause(1.27*(i-1));
    end
end
% The 7th cup is offset due to size limits in the gantry
if i == 7;
    configurePin(ard,Pulsey,'Unset');
    writeDigitalPin(ard,direcx,1);
    playTone(ard,Pulsex,1500,4.44);
    pause(4.44);
    configurePin(ard,Pulsex,'Unset');
    writeDigitalPin(ard,direcy,1);
    playTone(ard,Pulsey,1000,(1.27*(5)));
    pause(1.27*(5));
    configurePin(ard,Pulsey,'Unset');
    writeDigitalPin(ard,direcx,0);
    playTone(ard,Pulsex,1000,1.27);
    pause(1.27);
    configurePin(ard,Pulsey,'Unset');
```

```
    writeDigitalPin(ard,direcx,1);
    playTone(ard,Pulsex,1000,1.27);
    pause(1.27);
end

%Move down to cup
pos = 0.54;
ServoMotor(ard,pos);
disp('Moved Down to cup')
pause(1)
%Closes the lense on the cup
writePWMVoltage(ard, enablePin, 3); % Adjust voltage to control speed
writeDigitalPin(ard, in1Pin, 0); % Set direction (adjust for your setup)
writeDigitalPin(ard, in2Pin, 1); % Set direction (adjust for your setup)
pause(0.32); % Run the motor for 5 seconds
writePWMVoltage(ard, enablePin, 0);
pause(0.2)
disp('Lense Closed')
pause(2)
%Moves the mechanism up holding the cup
pos = 0;
ServoMotor(ard,pos);
disp('Moved Up')

% Returns back to the origin
OriginPlacement(ard);

end
```

Script 6 – Picking Up Cup Function.

```
function move_stepper_motor(motorpick,coord,ard)

    % Assign and configure the relevant variables and pins
    enablex = 'D45';
    direcx = 'D43';
    Terminalx = 'D39';
    enabley = 'D53';
    direcy = 'D51';
    Terminaly = 'D47';
    Pulsex = 'D41';
    Pulsey = 'D49';
    configurePin(ard,enablex,'DigitalOutput');
    configurePin(ard,direcx,'DigitalOutput');
    configurePin(ard,Terminalx,'DigitalInput');
    configurePin(ard,enabley,'DigitalOutput');
    configurePin(ard,direcy,'DigitalOutput');
    configurePin(ard,Terminaly,'DigitalInput');

    % Moves the X amount of coordinates including offset
    if motorpick == 1
        configurePin(ard,Pulsex,'Unset');
        writeDigitalPin(ard,direcy,1);
        distance = ((1.8/3)*coord)-0.7;
        playTone(ard,Pulsey,600,distance);
        pause(distance)
        disp('Moved x')
    end
    % Moves to the Y coordinates including offset due to hallf effect
    % sensor offset
    if motorpick == 2
        configurePin(ard,Pulsey,'Unset');
        writeDigitalPin(ard,direcx,1);
        distance2 = (0.7*coord);
        playTone(ard,Pulsex,600,distance2);
        pause(distance2)
        disp('Moved Y')
    end
end
```

Script 7 – Function to move the mechanism to magnet coordinates.